NAME

magic − file command's magic number file

DESCRIPTION

This manual page documents the format of the magic file as used by the **file**(1) command, version 3.33. The **file** command identifies the type of a file using, among other tests, a test for whether the file begins with a certain *magic number*. The file */dev/env/DJDIR/share/magic* specifies what magic numbers are to be tested for, what message to print if a particular magic number is found, and additional information to extract from the file.

Each line of the file specifies a test to be performed. A test compares the data starting at a particular offset in the file with a 1-byte, 2-byte, or 4-byte numeric value or a string. If the test succeeds, a message is printed. The line consists of the following fields:

offset A number specifying the offset, in bytes, into the file of the data which is to be tested.

type The type of the data to be tested. The possible values are:

- byte A one-byte value.
- short A two-byte value (on most systems) in this machine's native byte order.
- long A four-byte value (on most systems) in this machine's native byte order.
- string A string of bytes. The string type specification can be optionally followed by /[Bbc]*. The ''B'' flag compacts whitespace in the target, which must contain at least one whitespace character. If the magic has "n" consecutive blanks, the target needs at least "n" consecutive blanks to match. The ''b'' flag treats every blank in the target as an optional blank. Finally the ''c'' flag, specifies case insensitive matching: lowercase characters in the magic match both lower and upper case characters in the targer, whereas upper case characters in the magic, only much uppercase characters in the target.
- date A four-byte value interpreted as a unix date.
- beshort A two-byte value (on most systems) in big-endian byte order.
- belong A four-byte value (on most systems) in big-endian byte order.
- bedate A four-byte value (on most systems) in big-endian byte order, interpreted as a unix date.
- leshort A two-byte value (on most systems) in little-endian byte order.
- lelong A four-byte value (on most systems) in little-endian byte order.
- ledate A four-byte value (on most systems) in little-endian byte order, interpreted as a unix date.

The numeric types may optionally be followed by **&** and a numeric value, to specify that the value is to be AND'ed with the numeric value before any comparisons are done. Prepending a **u** to the type indicates that ordered comparisons should be unsigned.

test The value to be compared with the value from the file. If the type is numeric, this value is specified in C form; if it is a string, it is specified as a C string with the usual escapes permitted (e.g. \n for new-line).

Numeric values may be preceded by a character indicating the operation to be performed. It may be =, to specify that the value from the file must equal the specified value, <, to specify that the value from the file must be less than the specified value, >, to specify that the value from the file must be greater than the specified value, **&**, to specify that the value from the file must have set all of the bits that are set in the specified value, ^, to specify that the value from the file must have clear any of the bits that are set in the specified value, or **x**, to specify that any value will match. If the character is omitted, it is assumed to be =.

Numeric values are specified in C form; e.g. **13** is decimal, **013** is octal, and **0x13** is hexadecimal.

For string values, the byte string from the file must match the specified byte string. The operators =, < and > (but not **&**) can be applied to strings. The length used for matching is that of the string argument in the magic file. This means that a line can match any string, and then presumably print that string, by doing **>\0** (because all strings are greater than the null string).

message

The message to be printed if the comparison succeeds. If the string contains a **printf**(3S) format specification, the value from the file (with any specified masking performed) is printed using the message as the format string.

Some file formats contain additional information which is to be printed along with the file type. A line which begins with the character > indicates additional tests and messages to be printed. The number of > on the line indicates the level of the test; a line with no > at the beginning is considered to be at level 0. Each line at level *n*+**1** is under the control of the line at level *n* most closely preceding it in the magic file. If the test on a line at level *n* succeeds, the tests specified in all the subsequent lines at level *n*+**1** are performed, and the messages printed if the tests succeed. The next line at level *n* terminates this. If the first character following the last > is a **(** then the string after the parenthesis is interpreted as an indirect offset. That means that the number after the parenthesis is used as an offset in the file. The value at that offset is read, and is used again as an offset in the file. Indirect offsets are of the form: **((***x***[.[bslBSL]][+-][***y***]).** The value of *x* is used as an offset in the file. A byte, short or long is read at that offset depending on the **[bslBSL]** type specifier. The capitalized types interpret the number as a big endian value, whereas the small letter versions interpet the number as a little endian value. To that number the value of *y* is added and the result is used as an offset in the file. The default type if one is not specified is long.

Sometimes you do not know the exact offset as this depends on the length of preceding fields. You can specify an offset relative to the end of the last uplevel field (of course this may only be done for sublevel tests, i.e. test beginning with >). Such a relative offset is specified using **&** as a prefix to the offset.

BUGS

The formats *long*, *belong*, *lelong*, *short*, *beshort*, *leshort*, *date*, *bedate*, and *ledate* are system-dependent; perhaps they should be specified as a number of bytes (2B, 4B, etc), since the files being recognized typically come from a system on which the lengths are invariant.

There is (currently) no support for specified-endian data to be used in indirect offsets.

SEE ALSO

file(1) − the command that reads this file.